Homework # 14

In this homework, we will use an autoencoder to dimension reduce the MNIST dataest to a 2 *dimensional latent space and we'll compare the dimension reduction to what we get using PCA. We'll do the dimension reduction in two ways. Through an autoencoder that uses only the images, but not the accompanying image numbers labels, i.e.* $y = 2$ *if the image is of a* 2*. This is an unsupervised approach. And through an encoder with the addition of a prediction layer that does use the accompanying number labels.. This is a supervised approach.*

I've attached code solutions for both problems. Feel free to use the code, but use this opportunity to learn pytorch or improve your pytorch coding. Besides the attached code, ChatGPT and Claude (I think Claude is a bit better) are great resources for writing and understanding code in pytorch.

1. Use pytorch to fit an autoencoder to the MNIST data. If X is the MNIST data, rescale to $X/255$ to make all MNIST values lie between 0 and 1; this is just a convenient normalization. From the full MNIST dataset, leave out 5000 samples that will serve to test the autoencoder. In training the autoencoder, use all of the other 55000 samples.

    - The encoding layers should have dimensionality $784, 500, 250, k$ where $k = 2$ is the dimension of the latent space.
    - Use a ReLu between the encoder layers.
    - The decoder layers should have the reverse dimensionaliy.
    - Use a ReLu between the decoders layers, but then apply a sigmoid on the last layer to force the output values to lie between 0 and 1.
    - Train using batches of 100. This means that you should split the training dataset into groups of 100 and each update to the autoencoder should be based on 100 samples. Cycle through the batches to use the full 55000 samples. Each such cycle is referred to as an epoch. Train over several epochs, 10 for example.
    - Train using the mean squared error as the loss function. Code your own loss function, don't use pytorch's.

(a) Recall problem 2 of homework 4 in which we used PCA to dimension reduce MNIST to k dimensions. Here use $k = 2$. Letting x_i be the ith test image and $z_i = e(x_i) \in \mathbb{R}^2$ be the dimension reduction of x_i determined by the encoder, plot the z_i and color according to the number of the images. Do this using the encoder from the autoencoder and from the PCA. Compare the dimension reductions.

(b) Now we'll compare the decoders. Consider the first 10 images in the test dataset, $x_i \in \mathbb{R}^{784}$ for $i = 1, 2, 3, \ldots, 10$. Then $\tilde{x}_i = d(e(x_i)) \in \mathbb{R}^{784}$ where $d()$ and $e()$ are the decoder and encoder respectively. Visualize the $\tilde{x}_i$ by converting to a 28×28 matrix and using `imshow` or some other matrix display function:

```
from matplotlib import pyplot as plt

x = x.reshape(28,28)
plt.imshow(x)
```

Do this for the autoencoder and PCA. Which does a better job recovering the images?

2. Now we'll take a supervised approach to dimension reduction. Form a neural net by starting with the encoder architecture of problem 1, but then add a linear and softmax layer to the $k = 2$ dimenionsal output of the encoder:

```
import torch.nn as nn

nn.Linear(k, 10),
nn.Softmax()
```

Letting $f(x)$ be this neural net, $f(x) : \mathbb{R}^{784} \to \mathbb{R}^{10}$ and the soft max gaurantees $\sum_{i=1}^{10} f_i(x) = 1$ where $f_i(x)$ is the ith coordinate of $f(x)$. Letting y be the number shown in the image of sample x, we assume the model $f_i(x) = P(y = i \mid x)$, i.e. the probability that the image is of the number i.

(a) What loss function should be used to train this neural net? Train the neural net with $k = 2$ using the loss function you pick.

(b) Repeat 1a, using the encoder portion of the neural net to dimesion reduce the test dataset. Compare to the plots you generated in 1a.